Model predictive control

October 25, 2020

Stan Furrer 239499
Maxime Bonnesoeur 259009
Luis Carvalho 250184

1 Introduction

The goal of this project is to develop an MPC controller to fly a quadcopter. A linear MPC controller works with linear dynamics, convex constraints and a quadratic cost function.To reach the prerequisite, we will linearize our system at steady state position and divide it in four independent linear systems corresponding to the dimensions : x,y,z and yaw. Then, the constraints of each dimension will be written in a polytope way and a recursively feasible and stable set will be built. At this point, four independent MPC controllers for each dimension have been built. The goal being to make the nonlinear quadcopter follow a path, we will rewrite the MPC in a delta formulation. Afterwards, the controller will be updated so as to reject a constant disturbance d and to track a setpoint reference with no offset. Finally, a non linear MPC controller for the quadcopter will be developed using CASADI.

2 Linearization and Diagonalization

An linear MPC controller works with a linear dynamic. The system is defined by a state vector x containing 12 states describing the angular speed and angle in the mobile frame (attached to the quadcopter), and the velocities and positions in the fixed frame. The dynamic of the quadcopter is nonlinear and thus have to be linearized. We will see that the dynamics can be simplified into 4 linear sub-systems under certain conditions in order to utilize MPC techniques to make it follow a path.

2.1 Deliverable

- *Explain why this occurs, and whether this would occur at other steady-state conditions.*

Quadrotor trim is the definition of the drone flying at steady state. This means that all the drone velocity variables are fixed at a constant value. The drone is 'tuned' in a way that it conducts a steady flight and no torque is applied to the quadrotor's center of gravity. This implies that the drone acceleration components are all zero. This definition leads to $\dot{x_s} = f(x_s, u_s) = 0$ .The steady state of our system is $\vec{x_s} = \vec{0}$ and $\vec{u_s} = [0.7, 0.7, 0.7, 0.7]^T$. As the state vector is a null vector, the quadcopter motor only have to compensate for the gravitational force by having all the same thrust of 0.7. Intuitively, the robot can have any Cartesian position x,y,z and yaw in order to be in steady.state, while u_s remains the same.

Our model is then linearized around different trim conditions. Each trim condition is a steady state equilibrium, therefore, the state space system obtained for different operating conditions is stable.

The state-space matrices A(t), B(t), C(t), and D(t) of this linearized model represent the Jacobian matrices of the system. Since the steady state conditions are applied, the state space becomes a linear time invariant (LTI) state space system. The matrices A(t), B(t), C(t), and D(t) become time invariant. We would like to decompose our initial system in four independent systems that correspond to x, y, z and yaw. The initial state space of our model is based on world coordinate frame. This model have dependency between the rotor since moment can be created by the balance between the blades. From the equation of dependency given by T, we can apply a change of variable $u = T^1 v$ to create an independent system. In our original system, C is the unity matrix, D is the zero matrix and A is independent. Indeed, even if the matrix B is linearly independent, its rows 1, 2,3 and 9 are non-zeros and thus don't represent a canonical base of the R subspace. It involves that applying $\vec{u_s} = [1, 0, 0, 0]^T$ or $\vec{u_s} = [0, 0, 1, 0]^T$ as input, influences the same dimension. This can be seen in the matrix B below.

$$B_{old} = \begin{bmatrix} 0 & 0.56 & 0 & -0.56 \\ -0.56 & 0 & 0.56 & 0 \\ 0.73 & -0.73 & 0.73 & -0.73 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3.5 & 3.5 & 3.5 & 3.5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, B_{new} = \begin{bmatrix} 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.0667 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The change of variable allows to rewrite the new matrix BT^{-1} as B_{new}. This new matrix has only one component per row and per dimension in the non-zero rows. We can now divide our system in four independent sub-systems.

3 Design MPC Controllers for Each Sub-System

For each of the dimensions x, y, z and yaw, our goal is to design a recursively feasible, stabilizing MPC controller that can track step references.

3.1 Deliverable

- *Explanation of design procedure that ensures recursive constraint satisfaction*

  Each Sub-system MPC controllers has to satisfy the given constraints at any steps. We have to compute the invariant set.

  Our problem is convex. It has the benefits that any local optimal point is globally optimal.

  We will rewrite the constraint in a polytope way.

  - the constraint on the state should look like X = {x | Fx <= f}
  - the constraint on the input of the model should look like U = { u | Mu <= m }

  For each controller the constraints are a bit different but are well defined in the m-code delivered. These constraints define the Feasible set.

  Recursive constraint satisfaction is achieved by assuming feasible initial states and guaranteeing feasible state along the closed-loop trajectory. Ideally we would solve the MPC problem with an infinite horizon, but that is computationally intractable. We bound the tail of the infinite horizon cost from N to ∞ using LQR control law $u = K_{LQR}x$. Concretely, before $i = N$, the constraints are checked for every prediction in the finite horizon. Then for the prediction at i = N (producing terminal weight), it is the terminal set constraint that is checked by the solver.

- *Explanation of choice of tuning parameters. (e.g., Q, R, N, terminal components)*

  The main idea in LQR control design is to minimize the quadratic cost function of $int(x^TQx + u^TRu)dt$. It turns out that regardless of the values of Q and R, the cost function has a unique minimum that can be obtained by solving the Algebraic Riccati Equation. The parameters Q and R can be used as design parameters to penalize the state variables and the control signals. The larger these values are, the more these signals are penalized. Basically, choosing a large value for R is equivalent to try to stabilize the system with less (weighted) energy. On the other hand, choosing a small value for R is equivalent to not penalize the control signal. Similarly, choosing a large value for Q means trying to stabilize the system with the least possible changes in the states and a large Q implies less concern about the changes in the states. Since there is a trade-off between the two, we want to keep Q as I (identity matrix) and only alter R. Our design strategy is to penalize the actuators to ensure energy efficiency.

However, the actuators cannot be extremely penalized otherwise the steady-state will also be penalized. In order to choose R, we plot in addition the function u for two different values of R.

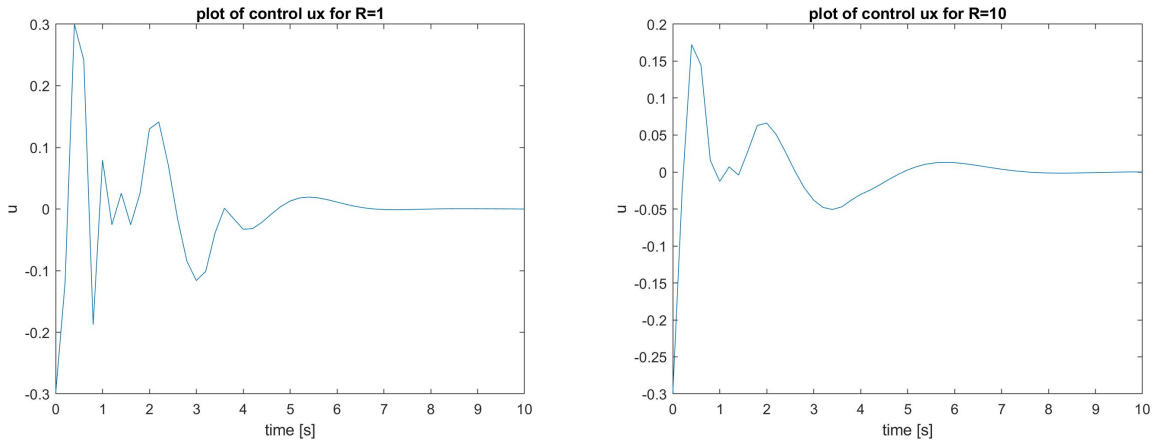


Figure 1: Function ux for R=1 and R=10. When penalizing the controller (R=10), the curve of ux is smoother and has less variance.

The plots show that penalizing the controller (R=10), brings smoothness to the controller u. By choosing R=10 the variance becomes smaller which is necessary regarding energy efficiency. Moreover, increasing R will maximize the terminal set. The values of the parameters are set equally amongst all the controllers (x,y,z and yaw).

The horizon has a strong impact on the computation cost (Q_f). Through a few experiences we found that N = 50 is a good trade off between time consuming and size of the feasible set (a demonstration of the influence of the parameters is shown in the section 4.1).

- *Plot of terminal invariant set for each of the dimensions, and explanation of how they were designed and tuning parameters used*

  The terminal set were obtained by computing the optimal (unconstrained) LQR controller to "fake" the infinite horizon. The terminal cost Q_f is as well extracted from the LQR solution.

  The computation of the terminal sets from an LQR controller depend on Q and R and not N. Only the terminal cost depends on N. The terminal set size increase as R.

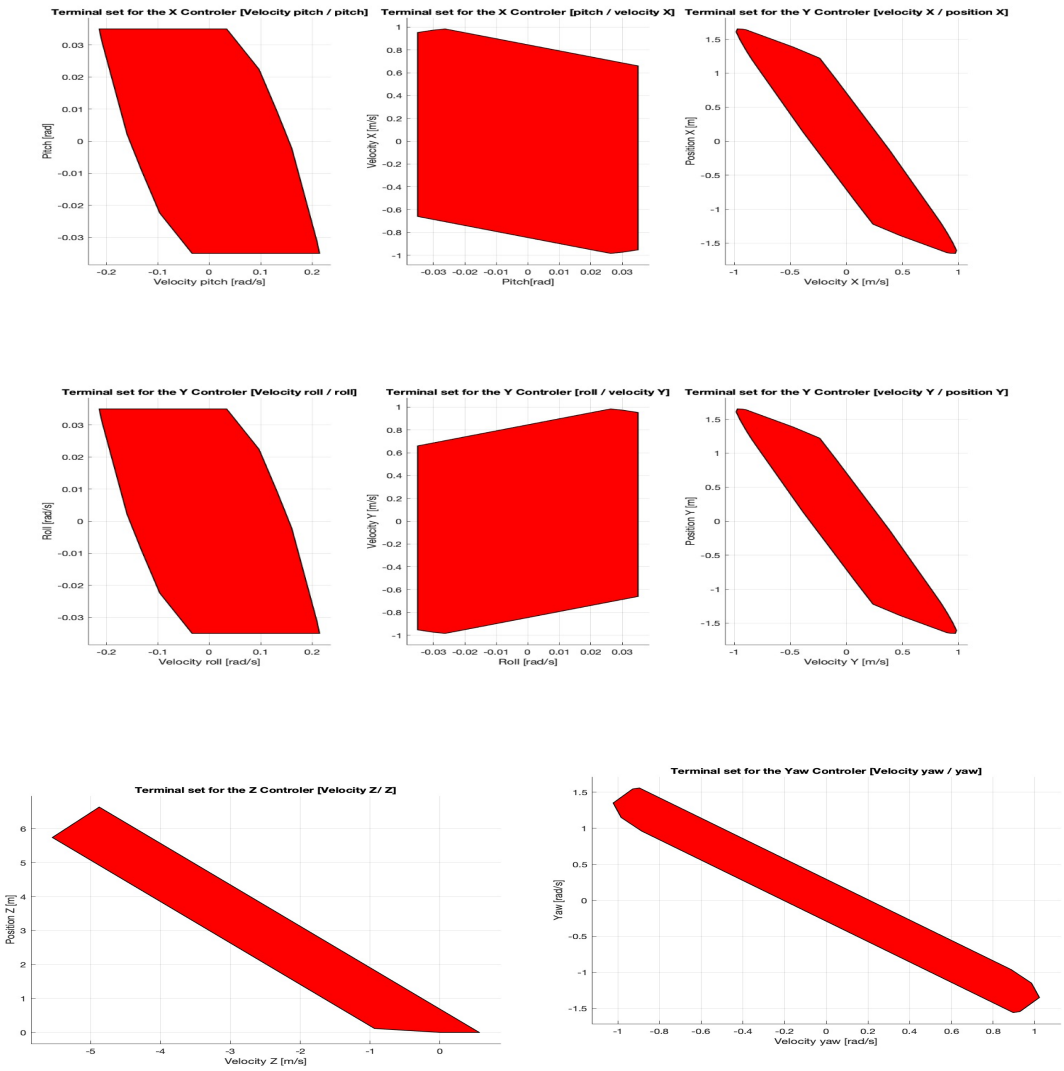


Figure 2: Maximum Invariant set

We observe in the figure 1 that each model allows the system to stabilize to origin with the above parameters Q, R and N starting 2 meters away (for x, y, z) or with an initial angle of 45 degrees for yaw. As expected we observe the convergence of each sub-system towards the origin for every dimension.

- *Plot for each dimension starting stationary at two meters from the origin (for x, y and z) or stationary at 45 degrees for yaw*

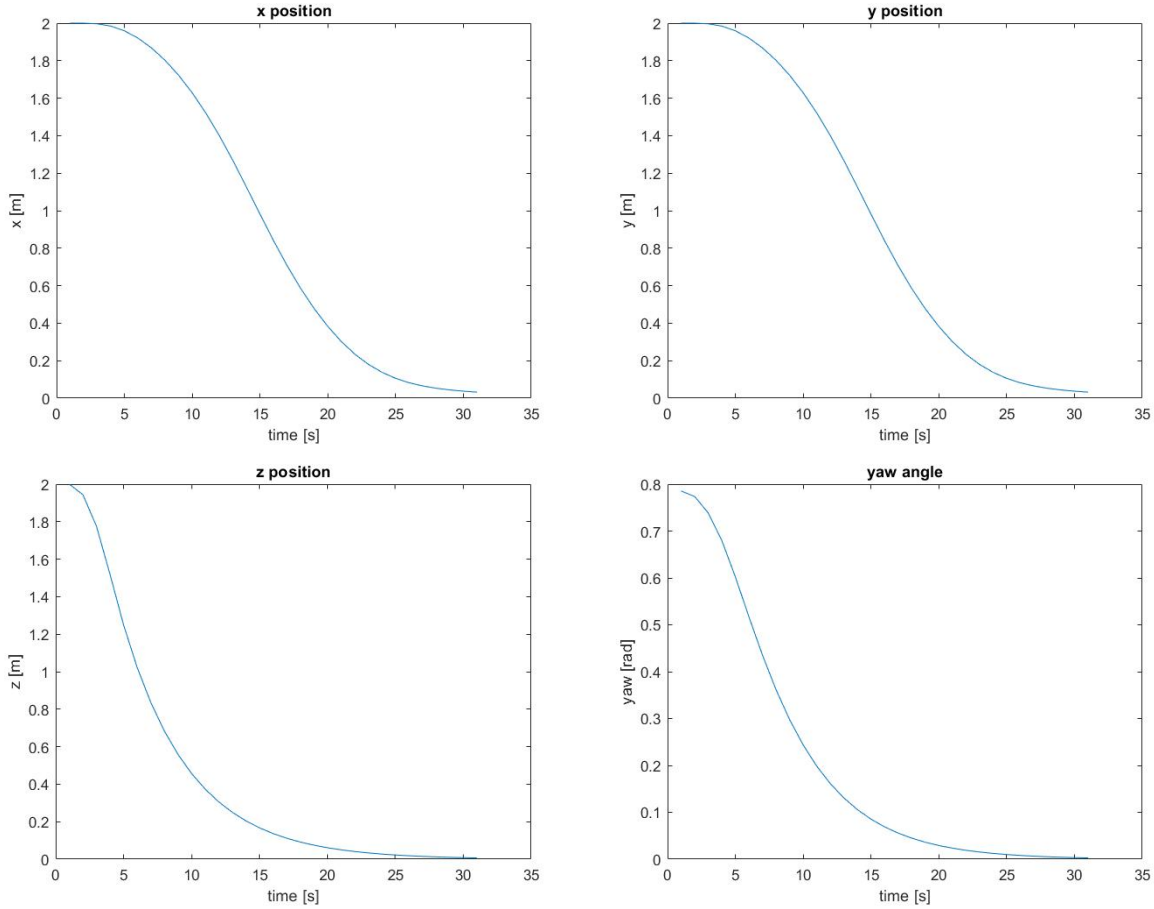


Figure 3: Plots of the controllers starting from 2 meters and stabilizing at the origin

3.2 Deliverable

- *Choice of tuning parameters*

  We kept the same parameters as discussed in 3.1

- *Explanation of your design procedure*

  Above, the controllers were stabilizing at the origin and now we are also going to be able to use them to track a constant reference r. The problem becomes a minimization of the input needed to reach the reference with the target steady-state x_s. The controllers maintain their feasibility and this target is achieved with the following equation:

$$\begin{aligned} \dot{x}_s &= Ax_s + Bu_s \\ Cx_s &= r \end{aligned} \tag{1}$$

  The next equations demonstrate the computation of the new steady-state by the optimizer using the objectives and constraints.

$$min(Cx_s - r)^T Q_s (Cx_s - r) \tag{2}$$

$$\text{s.t. } x_s = Ax_s + Bu_s \tag{3}$$

$$H_x x_s \leq k_x \tag{4}$$

$$H_u u_s \leq k_u$$

  It is not required to have a terminal set since we can drop the invariance, which means that we don't have any constraints neither on the positions nor the yaw angle. We can now update the control with the steady-state computed with the following equations:

$$\begin{aligned} &\min \sum_{i=0}^{N-1} \Delta x_i^T Q \Delta x_i + \Delta u_i^T R \Delta u_i + V_f\left(\Delta x_N\right) \\ &\text{s.t. } \Delta x_0 = \Delta x \\ &\Delta x_{i+1} = A\Delta x_i + B\Delta u_i \\ &H_x \Delta x_i \leq k_x - H_x x_{\mathrm{s}} \\ &H_u \Delta u_i \leq k_u - H_u u_s \\ &\Delta x_N \in \mathcal{X}_f \end{aligned} \tag{5}$$

  with $\Delta x_i = x_i - x_s, \Delta u_i = u_i - u_s$ and $\Delta x_N = x_N - x_s$

- *Plot for each dimension of the system starting at the origin and tracking a reference to -2 meters from the origin (for x, y and z) and to 45 degrees for yaw*

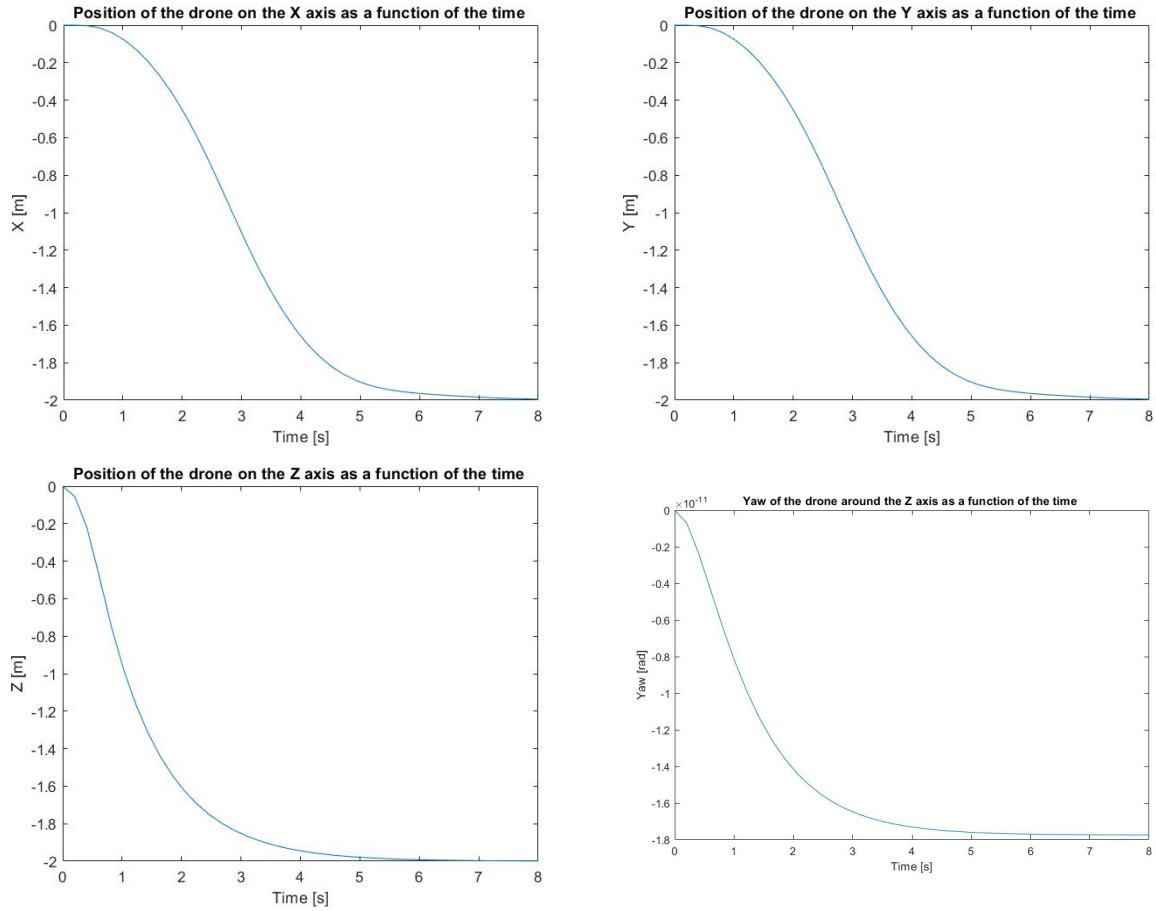
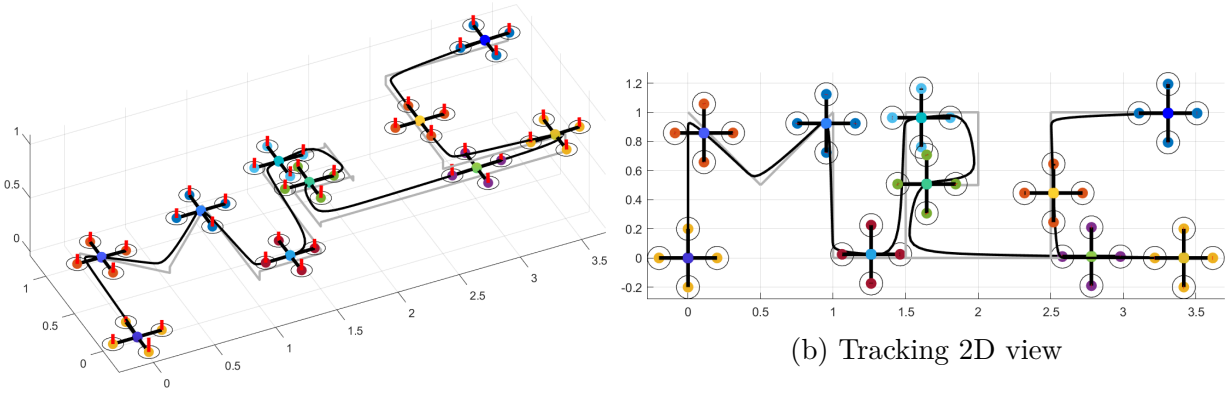


Figure 4: Plots of the controllers starting from the origin and stabilizing at the reference of -2 meters

4 Simulation with Nonlinear Quadcopter

4.1 Deliverable

- *A plot of your controllers successfully tracking the path using quad.plot(sim)*



(a) Tracking 3D view

(b) Tracking 2D view

Figure 5: Simulation with Non-linear quadcopter. The grey path is the path that we want to track. The black one is the path obtained thanks to our predictive control model

The two images above show how our drone follow a given path thanks to our four model predictive control applied. In the left result we can recognize the initials of the course (MPC). In the following part, we study the tuning of the parameters to see if our result corroborates our prediction in section 3.1.

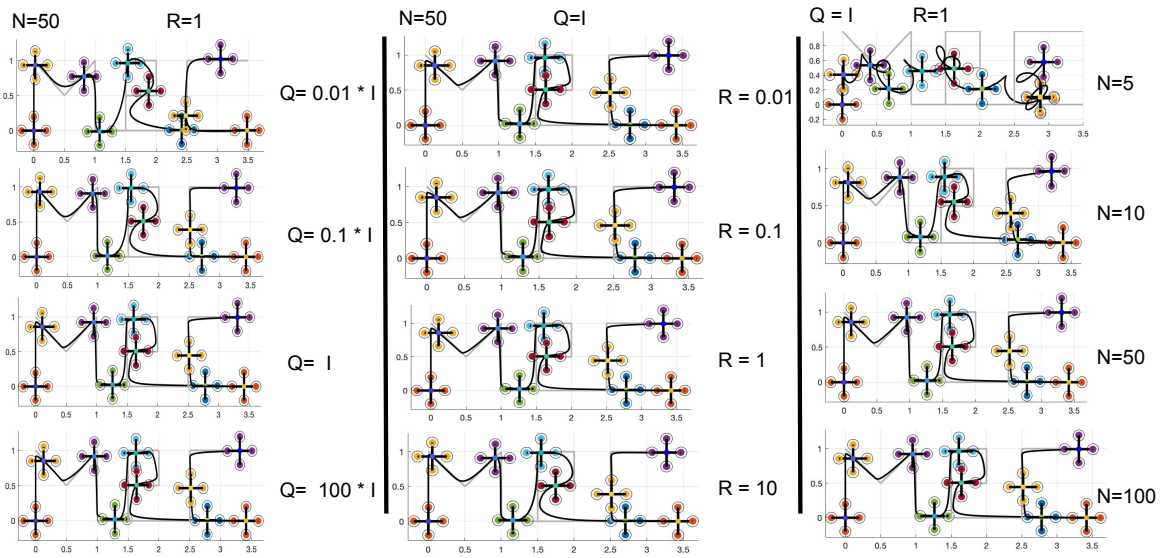


Figure 6: Influence of the parameters Q, R and N on the trajectory

From the figure 6, we can see the influence of the different parameters on the trajectory. The most noticeable change is the one on the horizon N. By taking an horizon too small like N = 5, the system, will not have a proper behavior. Once a good enough N is defined, in our case 50, the system will be stable and no further improvements will be obtained with a larger N.

As for the coefficient Q, we want our system to fit our desired trajectory with a high accuracy without breaking our constraints. From our results, it seems that taking a Q equal to the identity matrix provides satisfying results without being too constraining on the other parameters.

Finally, playing on the R parameter is acting on the penalization of the system. Hence, picking a small R, will give an accurate trajectory while augmenting it will smooth the trajectory. In our case, with the sharp corners of some letters such as P, picking an R equal to 10 is actually a good compromise between accuracy and smoothness of trajectory.

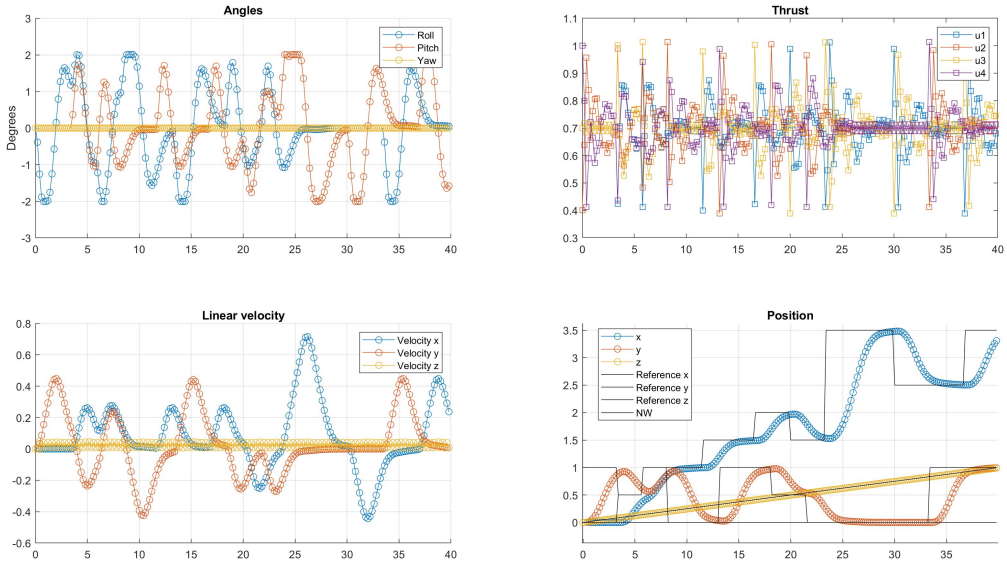


Figure 7: Simulation with Non-linear quadcopter.

Figure 6 shows the quadcopter tracking the reference over a given path with good precision. We notice that the controllers are always able to reach the target but tracking x and y position involves some latency. The latency depends on the controller outputs that are bounded. The tuning of R (control matrix) can be done in order to penalize less high value u_i, involving stiffer actuation. However, there is a trade off since the cost of transport will be higher, thus the robot consume more energy.The small imperfections are due to the linearization.

5 Offset-Free Tracking

5.1 Deliverable

- *Explanation of your design procedure and choice of tuning parameters*

  In this part, we will implement the capability of the controller to properly work even with any additional mass on the quadcopter. The new equation on the z-direction and the objective to be minimized becomes as follows:

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} + B\mathbf{d}$$

$$\text{objective} = u_s^2$$

  We obtain the target steady-state x_s and the reference with the next pair of equations:

$$\begin{aligned} \mathbf{x}_{s+} &= A\mathbf{x}_s + B\mathbf{u}_s + B\mathbf{d} \\ ref &= C\mathbf{x}_s + D + \mathbf{d} \end{aligned}$$

  The vector $\mathbf{d}$ is the disturbance and for now it is unknown. To calculate it, first we need to define $\overline{x_+}$ with the equations below:

$$\bar{B} = \begin{bmatrix} B \\ 0 \end{bmatrix} \quad \bar{A} = \begin{bmatrix} A & B \\ \text{eye}(2) & 1 \end{bmatrix} \quad \bar{C} = \begin{bmatrix} C & \text{ones } (2,1) \end{bmatrix}$$

$$\overline{\mathbf{x}_+} = \bar{A} * \overline{\mathbf{x}} + \bar{B}\mathbf{u} + L * (\bar{C} * \overline{\mathbf{x}} - ref)$$

  Above, we still have L to determine. Knowing that the observer of the system must be asymptotically stable, it means that the matrix **A-LC** must have all its eigenvalues inside the unit circle. Which brings us to the following solution:

$$L = -\text{ place}\left(\bar{A}', \bar{C}', [\lambda_1, \lambda_2, \lambda_3]\right)'$$

  With L calculated above, we can determine $\overline{x_+}$ and then obtain $\mathbf{d}$ from the difference between $\mathbf{x}$ and $\overline{x_+}$. Note that small eigenvalues $\lambda_1, \lambda_2, \lambda_3$ will result in a low initial overshoot, which also means that it will reduce the learning rate.

- *Plot showing that the z-controller achieves offset-free tracking*

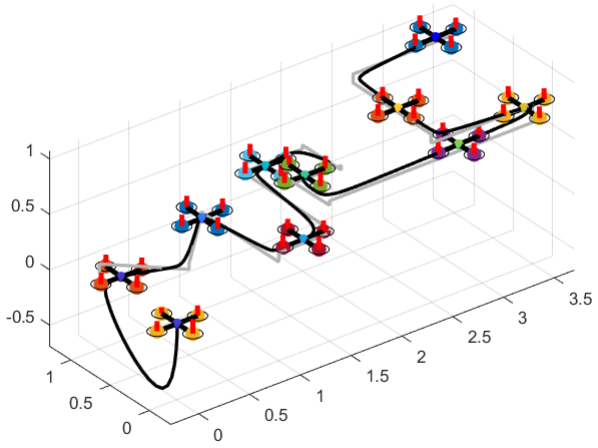


Figure 8: Simulation with Non-linear quadcopter.This time the mass of the quadcopter changes. The grey path is the path that we want to track. In dark in the path obtain thanks to our predictive control model

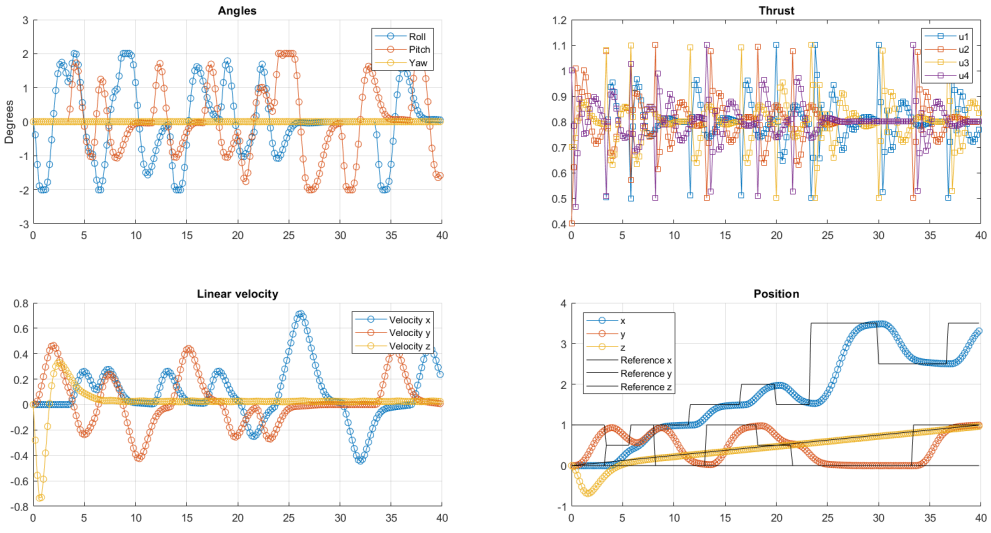


Figure 9: Simulation visualization of the different dimensions with a sampling time of 200ms

We saw in Figure 8 that at the beginning of the drone's course there is a great fall before picking up the correct path. This perfectly illustrates the unexpected extra mass on it that takes a certain time to be adjusted. This time depends on the sampling time, for example, the next images are the same scenario but with a sampling time of 100ms instead of 200ms

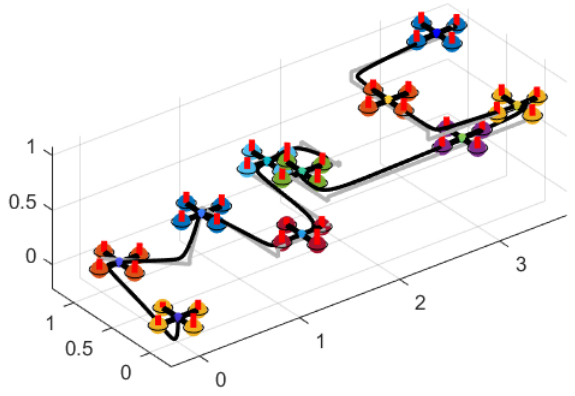


Figure 10: Simulation with Non-linear quadcopter.This time the mass of the quadcopter changes. The grey path is the path that we want to track. In dark in the path obtain thanks to our predictive control model

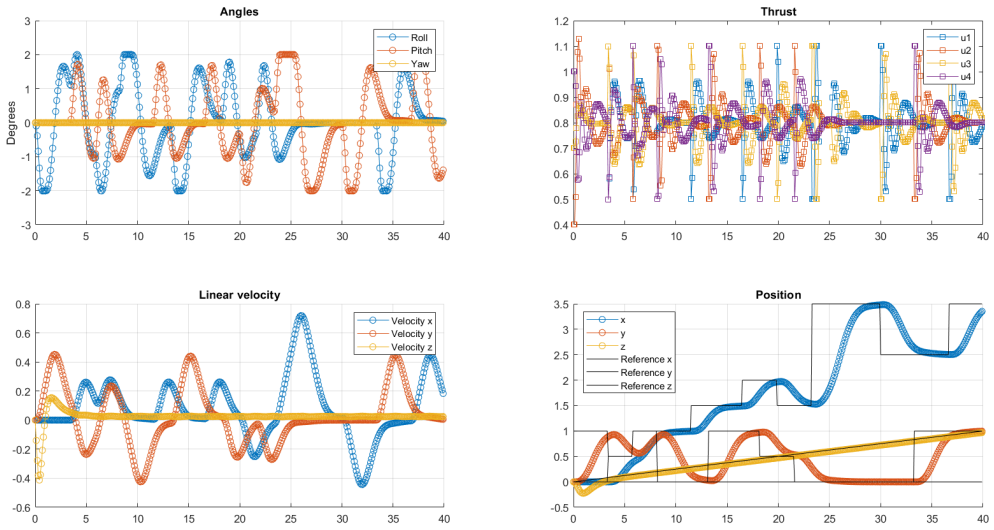


Figure 11: Simulation visualization of the different dimensions with a sampling time of 100ms (from left to right: angle, thrust, linear velocity and position

As we can see above, halving the sampling time, greatly reduced the initial slope because the system ca react faster to the unexpected mass.

6 Nonlinear MPC (Bonus)

6.1 Deliverable

- *Explanation of your design procedure and choice of tuning parameters*

Our design procedure was to define an objective function to minimize based on the speed, position and inputs of the robot. The objective function was then constrained on the inputs ($0 <= U <= 1.5$) and on the future set of outputs ($X_{k+1} == F(X_k, \dot{X}_k)$) where F is a predictor that is estimating the next state based on the Runge-Kutta equation and $\dot{X} = f(X, U)$ where f is the dynamic equation of the system).

Concerning the tuning of our parameters, we decided to use the same horizon N=50 for our tests. This resulted into a long processing time but gave us better results than the previous model that we implemented.
As with our previous model, we used an objective function on the input on the x y, and z position of our system but also on the velocity of the x,y z directions. Indeed, without considering the velocity in our previous model, there was some overshoot when the objective was changing. This was probably due to the drone that was going too fast and could not slow down enough to follow the trajectory.
Finally, in order to be more accurate in the z axis, we decided to have a weight five times bigger in the objective function for the z axis.

- *Explanation of why your nonlinear controller is working better than your linear one.*

At the beginning of the exercise, we made an approximation on the angle α and β to perform a linearization of our system. This linearization was necessary to be able to divide our system into several subsystems. However, this approximation and the linearization induced a small error in our global system. Hence, by using the whole system without doing any approximation gave us better results.

- *Plots showing the performance of your controller.*

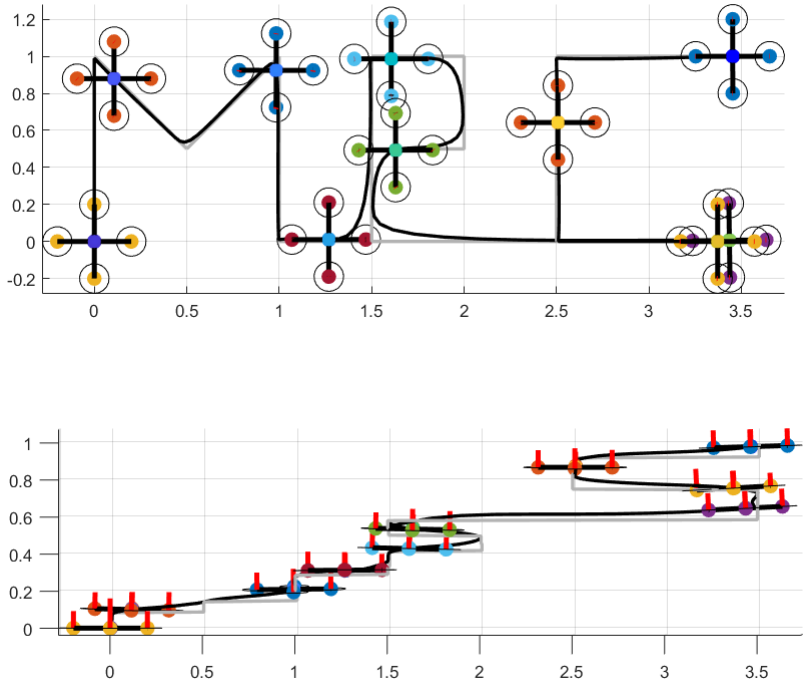


Figure 12: The images above show the controller's performance by showing the view from above and from the side and we can see how well the lines overlap

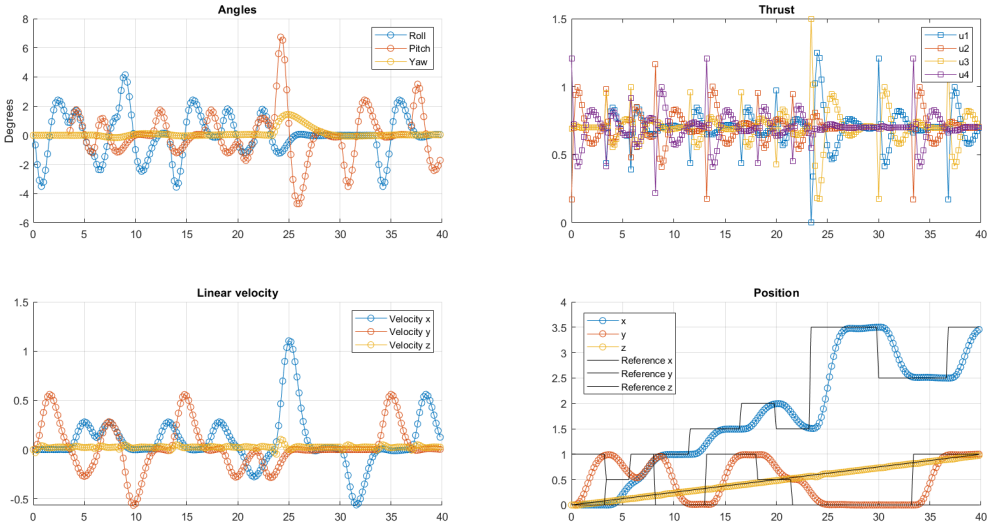


Figure 13: Visualisation of the different dimensions Non-linear MPC